

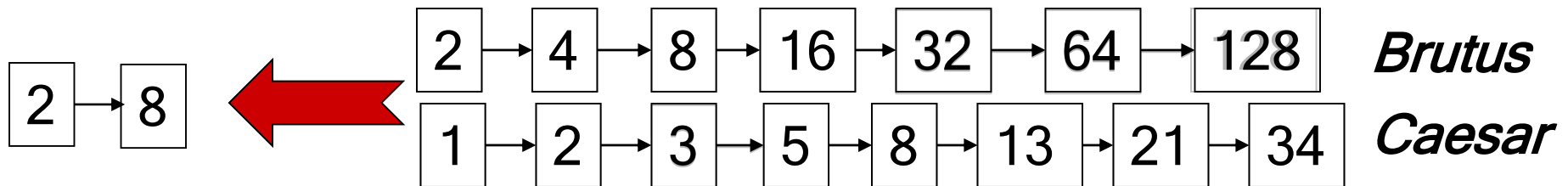
# Web Engineering

## Lecture 5

**Faster Postings Merges  
Skip Pointers & Skip Lists**

# Remember: Postings merging

- Walk through the two postings simultaneously, in time linear in the total number of postings entries



If the list lengths are  $x$  and  $y$ , the merge takes  $O(x+y)$  operations.

Can we do better? (next slide)

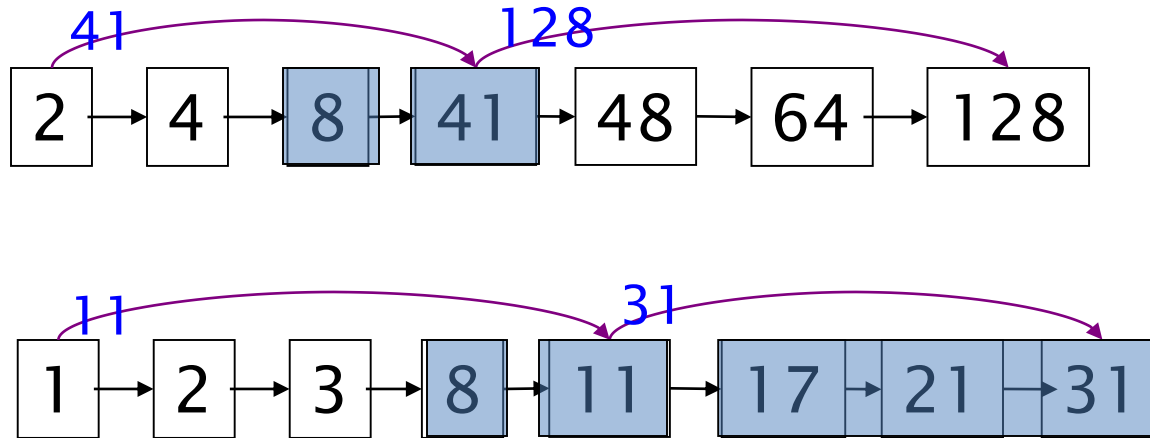
# Faster merging with skip pointers

---

Postings list intersection could be done in sublinear time, if the index isn't changing too fast:

- Using a **skip list** by *augmenting* postings lists with skip pointers (at indexing time).
- **Skip pointers** are effectively shortcuts to skip postings that ***will not figure in the search results***.
  - How to do efficient merging?
  - Where do we place skip pointers?

# Query processing with skip pointers



Suppose we've stepped through the lists until we process 8 on each list. We match it and advance.

We then have 41 and 11 on the lower. 11 is smaller.

But the skip successor of 11 on the lower list is 31, so we can skip ahead past the intervening postings.

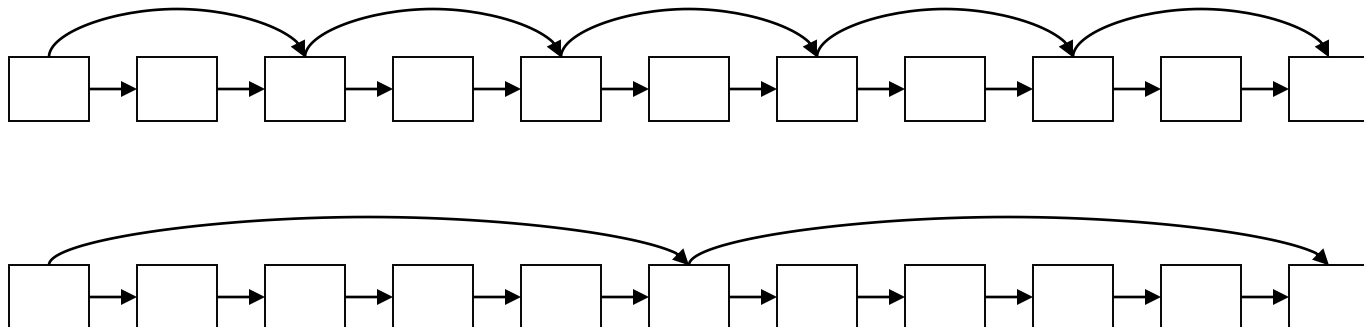
# skip pointers

INTERSECTWITHSKIPS( $p_1, p_2$ )

```
1  answer  $\leftarrow \{\}$ 
2  while  $p_1 \neq \text{NIL}$  and  $p_2 \neq \text{NIL}$ 
3  do if  $\text{docID}(p_1) = \text{docID}(p_2)$ 
4      then  $\text{ADD}(\text{answer}, \text{docID}(p_1))$ 
5           $p_1 \leftarrow \text{next}(p_1)$ 
6           $p_2 \leftarrow \text{next}(p_2)$ 
7  else if  $\text{docID}(p_1) < \text{docID}(p_2)$ 
8      then if  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
9          then while  $\text{hasSkip}(p_1)$  and  $(\text{docID}(\text{skip}(p_1)) \leq \text{docID}(p_2))$ 
10             do  $p_1 \leftarrow \text{skip}(p_1)$ 
11             else  $p_1 \leftarrow \text{next}(p_1)$ 
12 else if  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
13     then while  $\text{hasSkip}(p_2)$  and  $(\text{docID}(\text{skip}(p_2)) \leq \text{docID}(p_1))$ 
14         do  $p_2 \leftarrow \text{skip}(p_2)$ 
15         else  $p_2 \leftarrow \text{next}(p_2)$ 
16 return answer
```

# Where do we place skips?

- Tradeoff:
  - More skips  $\rightarrow$  shorter skip spans  $\Rightarrow$  more likely to skip. But lots of comparisons to skip pointers.
  - Fewer skips  $\rightarrow$  few pointer comparison, but then long skip spans  $\Rightarrow$  few successful skips.



# Placing skips

---


- Simple heuristic: for postings of length  $L$ , use  $\sqrt{L}$  evenly-spaced skip pointers.
- This ignores the distribution of query terms.
- Easy if the index is relatively static; harder if *a postings list* keeps changing because of updates.

# Complete example of skip pointers

- Suppose we have a two-word query:
  - For one term the postings list consists of the following 9 entries: [2,10,16,18,22,32,81,122,157].
  - For the other term, it is the one entry posting list:[32].

How many comparisons would be done to intersect the two postings lists using skip pointers?

- Using skip pointers of length 3 for the longer list and of length 1 for the shorter list:
    - The following comparisons will be made:
      1. 2 & 32
      2. 18 & 32
      3. 81 & 32
      4. 22 & 32
      5. 32 & 32
- Number of comparisons = 5



How many comparisons without skip pointers?





# **PHRASE QUERIES AND POSITIONAL INDEXES**

# Phrase queries

---

- Want to be able to answer queries such as “*stanford university*” – as a phrase
- Thus the sentence “*I went to university at Stanford*” is not a match.
  - Most recent search engines support a double quotes syntax (“stanford university”) for phrase queries.
- For this, it no longer suffices to store only *<term : docs>* entries

# A first attempt: Biword indexes

---

- Index every consecutive pair of terms in the text as a phrase
- For example the text “Friends, Romans, Countrymen” would generate the biwords
  - *friends romans*
  - *romans countrymen*
- Each of these biwords is now a dictionary term
- Two-word phrase query-processing is now immediate.

# Longer phrase queries

---

- Longer phrases can be processed by breaking them down:
- ***stanford university palo alto*** can be broken into the Boolean query on biwords:  
***stanford university AND university palo AND palo alto***

Without the docs, we cannot verify that the docs matching the above Boolean query do contain the phrase.



Can have false positives!

# Issues for biword indexes

---

- False positives, as noted before
- Index blowup due to bigger dictionary
  - Infeasible for more than biwords, big even for them
- we also need to have an index of single-word terms.
- Biword indexes are not the standard solution (for all biwords) but can be part of a compound strategy

# Solution 2: Positional indexes

---

- Positional indexes are a more efficient alternative to biword indexes.
- Postings lists in a **nonpositional** index: **each posting is just a docID**
- Postings lists in a **positional** index: **each posting is a docID and a list of positions**
- Example query: *“to1 be2 or3 not4 to5 be6”*

### Document 1

The bright blue butterfly hangs on the breeze.

### Document 2

It's best to forget the great sky and to retire from every wind.

### Document 3

Under blue sky, in bright sunlight, one need not search around.

### Query

Q- "blue sky"

### Inverted index

ID	Term	Document : position
1	best	2 : 3
2	blue	1 : 3, 3 : 2
3	bright	1 : 2, 3 : 5
4	butterfly	1 : 4
5	breeze	1 : 8
6	forget	2 : 5
7	great	2 : 7
8	hangs	1 : 5
9	needs	3 : 8
10	retire	2 : 11
11	search	3 : 10
12	sky	2 : 8, 3 : 3
13	wind	2 : 14

### Match on sequential terms

blue - 3 : 2  
sky - 3 : 3

### Search object

Document reference	Relevance
3	100%

# Positional index example

---

<*be*: 993427;

*1*: 7, 18, 33, 72, 86, 231;

*2*: 3, 149;

*4*: 17, 191, 291, 430, 434;

*5*: 363, 367, ...>

- For phrase queries, we use a merge algorithm recursively at the document level
- But we now need to deal with more than just equality



# Processing a phrase query

---

- Extract inverted index entries for each distinct term: ***to, be, or, not.***
- Merge their *doc:position* lists to enumerate all positions with “***to be or not to be***”.
  - ***to:***
    - 2:1,17,74,222,551; 4:8,16,190,429,433; 7:13,23,191; ...
  - ***be:***
    - 1:17,19; 4:17,191,291,430,434; 5:14,19,101; ...
- Same general method for proximity searches

# Processing a phrase query

TO, 993427:

1:  $\langle 7, 18, 33, 72, 86, 231 \rangle$ ;  
2:  $\langle 1, 17, 74, 222, 255 \rangle$ ;  
4:  $\langle 8, 16, 190, 429, 433 \rangle$ ;  
5:  $\langle 363, 367 \rangle$ ;  
7:  $\langle 13, 23, 191 \rangle$ ; ...

BE, 178239:

1:  $\langle 17, 25 \rangle$ ;  
4:  $\langle 17, 191, 291, 430, 434 \rangle$ ;  
5:  $\langle 14, 19, 101 \rangle$ ; ...

Document 4 is a match!

- 1) look for documents that contain both terms.
- 2) Then, look for places in the lists where there is an occurrence of *be* with a token index one higher than a position of *to*,
- 3) Then look for another occurrence of each word with token index 4 higher than the first occurrence.

# Proximity queries

---

- LIMIT! /3 STATUTE /3 FEDERAL /2 TORT
  - / $k$  means “within  $k$  words of”.
- Clearly, positional indexes can be used for such queries; biword indexes cannot.
- For example: employment /4 place
- Find all documents that contain employment and place within 4 words of each other.
- *Employment agencies that place healthcare workers are seeing growth* is a hit.
- *Employment agencies that have learned to adapt now place healthcare workers* is not a hit.

# Combination schemes

---

- These two approaches can be profitably combined
  - For particular phrases ( *“Michael Jackson”*, *“Britney Spears”* ) it is inefficient to keep on merging positional postings lists
    - Even more so for phrases like *“The Who”*
- Williams et al. (2004) evaluate a more sophisticated mixed indexing scheme
  - A typical web query mixture was executed in  $\frac{1}{4}$  of the time of using just a positional index
  - It required 26% more space than having a positional index alone

# Midterm Exam Notes

---

- All lectures from 1 to 5 are included
- 3 Questions [15 marks: ASU , 100 marks UEL]
  - MCQ [10 marks]
  - Replace with Key term [2.5 marks]
  - Solve a problem with skip pointers [2.5 marks]
- Answer in the same exam sheet

Lecture exercises are not included in exam

---

*Thank  
You*